Timer0 Interrupts

ECE 376 Embedded Systems

Jake Glower - Lecture #20

Please visit Bison Academy for corresponding lecture notes, homework sets, and solutions

Timer0 Interrupts

- Timer interrupts are pretty useful: the PIC18F4620 has four:

Interrupt	Description	Input	Conditions	Enable	Flag
Timer 0	Trigger after N events N = 1 .. 2^24 100ns to 1.67 sec	RA4: TOCS = 1 OSC/4: TOCS = 0	N = (PS)(Y) T0CON = 0x88: PS = 1 T0CON = 0x87: PS = 256 TMR0 = -Y	TMR0ON = 1 TMR0IE = 1 TMR0IP = 1 PEIE = 1	TMR0IF
Timer 1	Trigger after N events N = 1 .. 2^19 100ns to 0.52 sec	RC0 TMR1CS = 1 OSC/4 TMR1CS = 0	N = (PS)(Y) T1CON = 0x81: PS = 1 T1CON = 0xB1: PS = 8 TMR1 = -Y	TMR1ON = 1 TMR1IE = 1 TMR1IP = 1 PEIE = 1	TMR1IF
Timer2	Interupt every N clocks N = 1 .. 65,535 100ns to 6.55ms	OSC/4	N = A * B * C A = 1..16 (T2CON 3:6) B = 1..256 (PR2) C = 1, 4, 16 (T2CON 0:1)	T2E = 1 TMR2IE = 1 PEIE = 1	TMR2IF
Timer 3	Trigger after N events N = 1 .. 219 100ns to 0.52 sec	RC1 TMR3CS = 1 OSC/4 TMR3CS = 0	N = (PS)(Y) T3CON = 0x81: PS = 1 T3CON = 0xB1: PS = 8 TMR3 = -Y	TMR3ON = 1 TMR3IE = 1 TMR3IP = 1 PEIE = 1	TMR3IF

Timer0 Interrupt

- Similar to Timer1 & Timer3

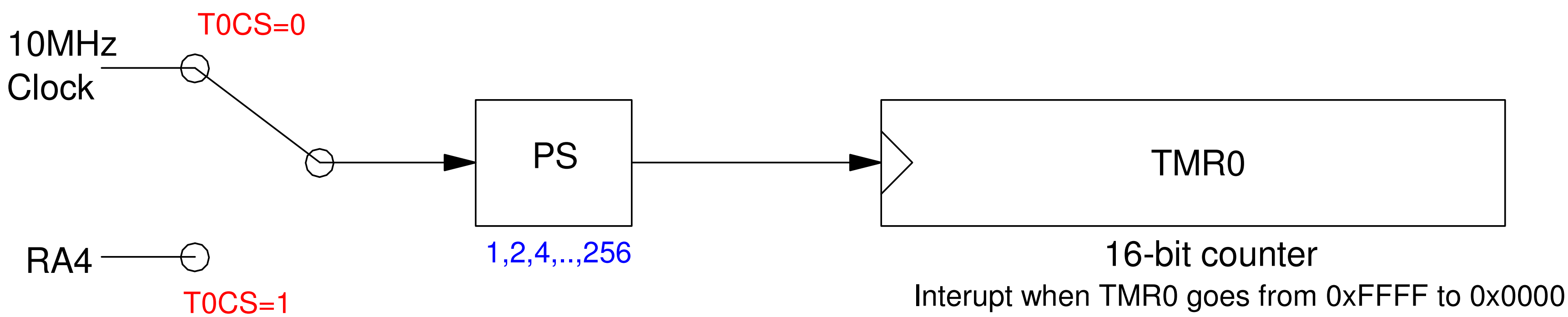
Input can be either

- External input, RA4 (T0CS = 0), or
- 10MHz clock (T0CS = 1)

The input goes to a divider (pre-scalar)

- Timer0: PS = {1, 2, 4, 8, 16, 32, 64, 128, 256}
- Timer1 & 3: PS = {1, 2, 4, 8}

Result goes to a 16-bit counter



Timer 0 Interrupt

What triggers the interrupt is TMR0 going to 0x0000

```
 TMR0
0xFFFD   -3
0xFFFE   -2
0xFFFF   -1
0x0000    0   Interrupt Triggered
0x0001   +1
0x0002   +2
```

Net Result

- N = 1 to 2^24

External Events: Count every 7th Button Push

One use of TIMER0 is to count every Nth rising edge. To do this

- Set the input to RA4 (T0CS = 1)
- Set up TMR0 = -7

  after seven rising edges, TMR0 = 0, which triggers the interrupt

- Inside the interrupt service routine, reset TMR0 = -7

  the next interrupt will be 7 rising edges later.

T0_Ext.c: Count every 7th Button Push

Interrupt Service Routine
Count every 7th edge

```
void interrupt IntServe(void)
{
   if (TMR0IF) {
      TMR0 = -7;
      N += 1;
      TMR0IF = 0;
      }
   }
```

Interrupt Initialization
External Input, PS = 1

```
// PS = 1
   T0CON = 0x88;

// External Input
   T0CS = 1;

// Enable Timer0
   TMR0ON = 1;
   TMR0IE = 1;
   TMR0IP = 1;
   PEIE = 1;
   GIE = 1;
```

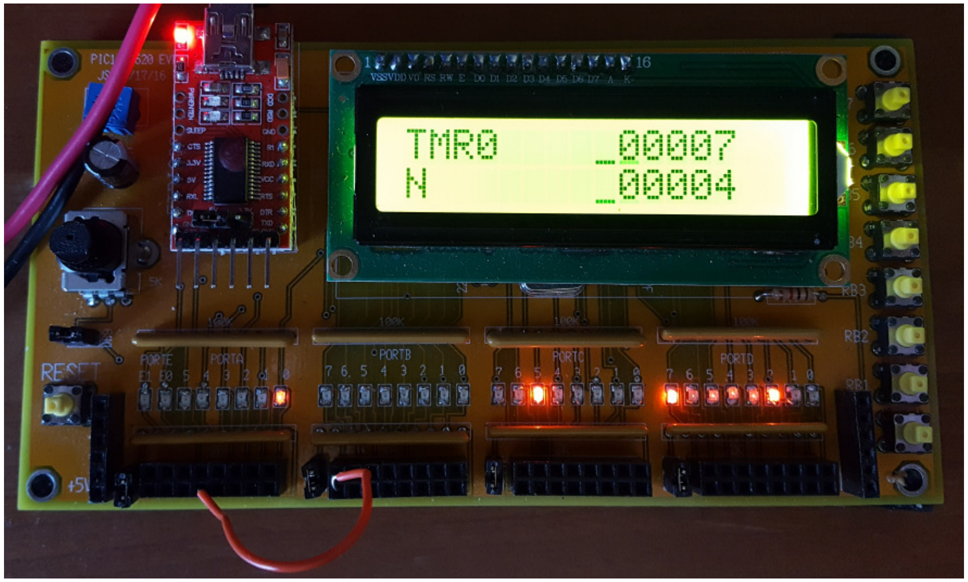
T0_Ext.c: Count every 7th Button Push

- What triggers the interrupt is TMR0 going 0x0000.
- If you do nothing, TMR0 won't return to 0x0000 for another 65,536 counts
- If you don't initialize TMR0, the first interrupt may not happen for 65,535 events

Main Routine

```
while(1) {
  LCD_Move(0,8);
  LCD_Out(-TMR0, 5, 0);
  LCD_Move(1,8);
  LCD_Out(N, 5, 0);
  }
```

Result



Timer0: Default Rate with PS = 1:

- Change T0CS = 0

  Counts clocks

- Interrupts every N clocks

  1 < N < 2^24

  Similar to Timer2

Default with PS = 1 is 65,536

- TMR0 = 0x0000 triggers the interrupt
- This won't happen again for another 65,536 (2^16) clocks

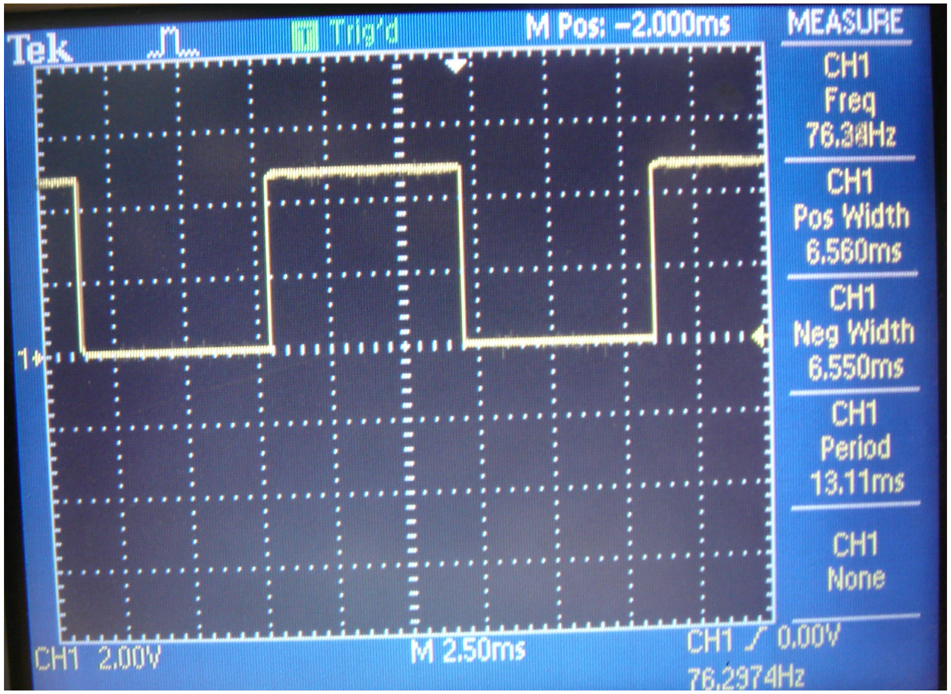
Timer0: Default Rate with PS = 1:

- Interrupts every 6.5536ms

Interrupt Service Routine

```
void interrupt IntServe(void)
{
   if (TMR0IF) {
      RC0 = !RC0;
      TMR0IF = 0;
      }
   }
```

Result



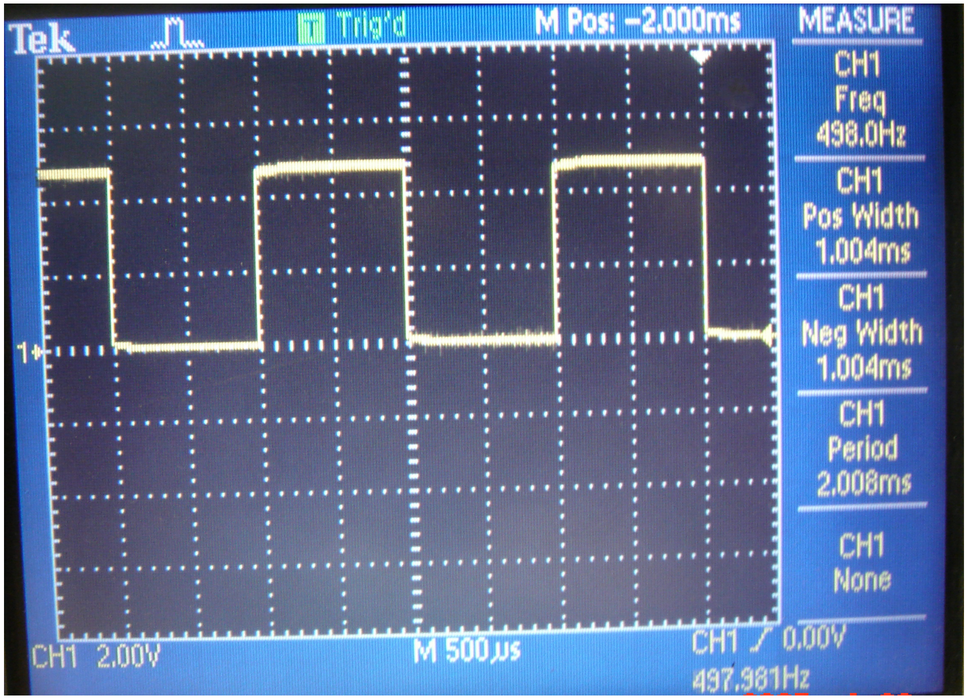
Timer0 Interrupt every 1ms

- You have to set up the next interrupt each time you interrupt
- Different than Timer2

Interrupt Service Routine

```
void interrupt IntServe(void)
{
   if (TMR0IF) {
      TMR0 = -10000;
      RC0 = !RC0;
      TMR0IF = 0;
      }
   }
```

Result



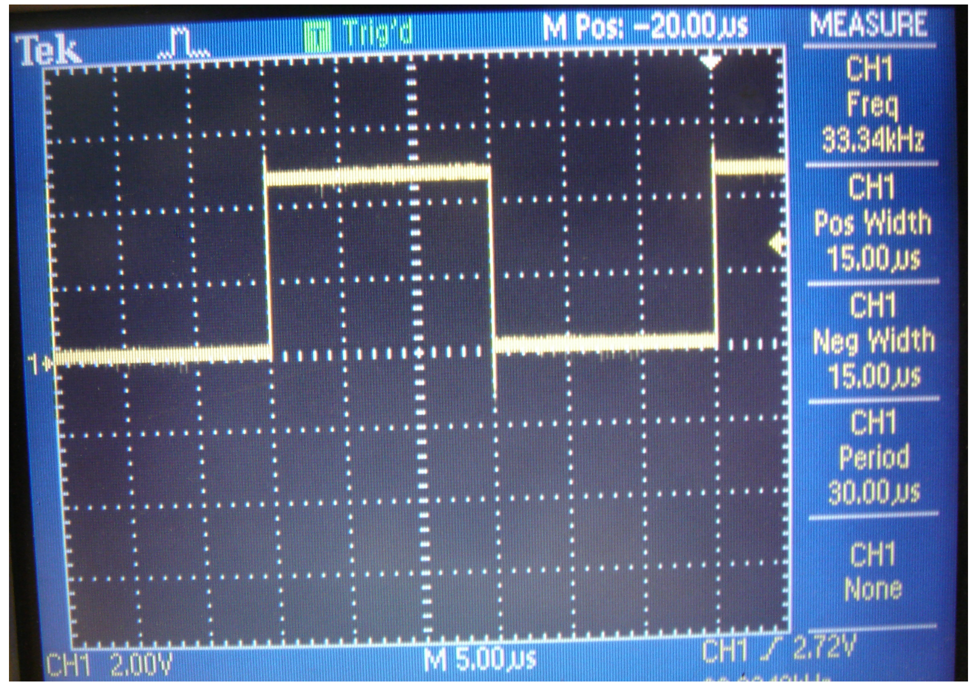
Note: The timing is off by about 50 clocks

- The time it takes to trigger the interrupt

Interrupt Service Routine
N = 100

```
void interrupt IntServe(void)
{
   if (TMR0IF) {
      TMR0 = -100;
      RC0 = !RC0;
      TMR0IF = 0;
      }
   }
```

Result
N is actually 150



Playing Note D5: 587.33Hz

- $N = \left(\frac{10{,}000{,}000}{2 \cdot Hz} \right) = 8513.1017$
- This is *way* easier than Timer2

Interrupt Service Routine

```
// Global Variable
unsigned int N = 8513 - 50;

void interrupt IntServe(void)
{
   if (TMR0IF) {
      TMR0 = -N;
      RC0 = !RC0;
      TMR0IF = 0;
      }
   }
```

Result

Measuring Time to 100ns

With Timer0, you can measure time to 100ns

- TIME: 32-bit variable (long integer)
- TMR0: Low 16-bits (100ns resolution)
- High 16-bits: Increment every Timer0 interrupt

TIME 32-bit variable	
high 16 bits	low 16 bits
TIME(31:16)	TMR0

Measuring Time to 100ns

Interrupt Service Routine

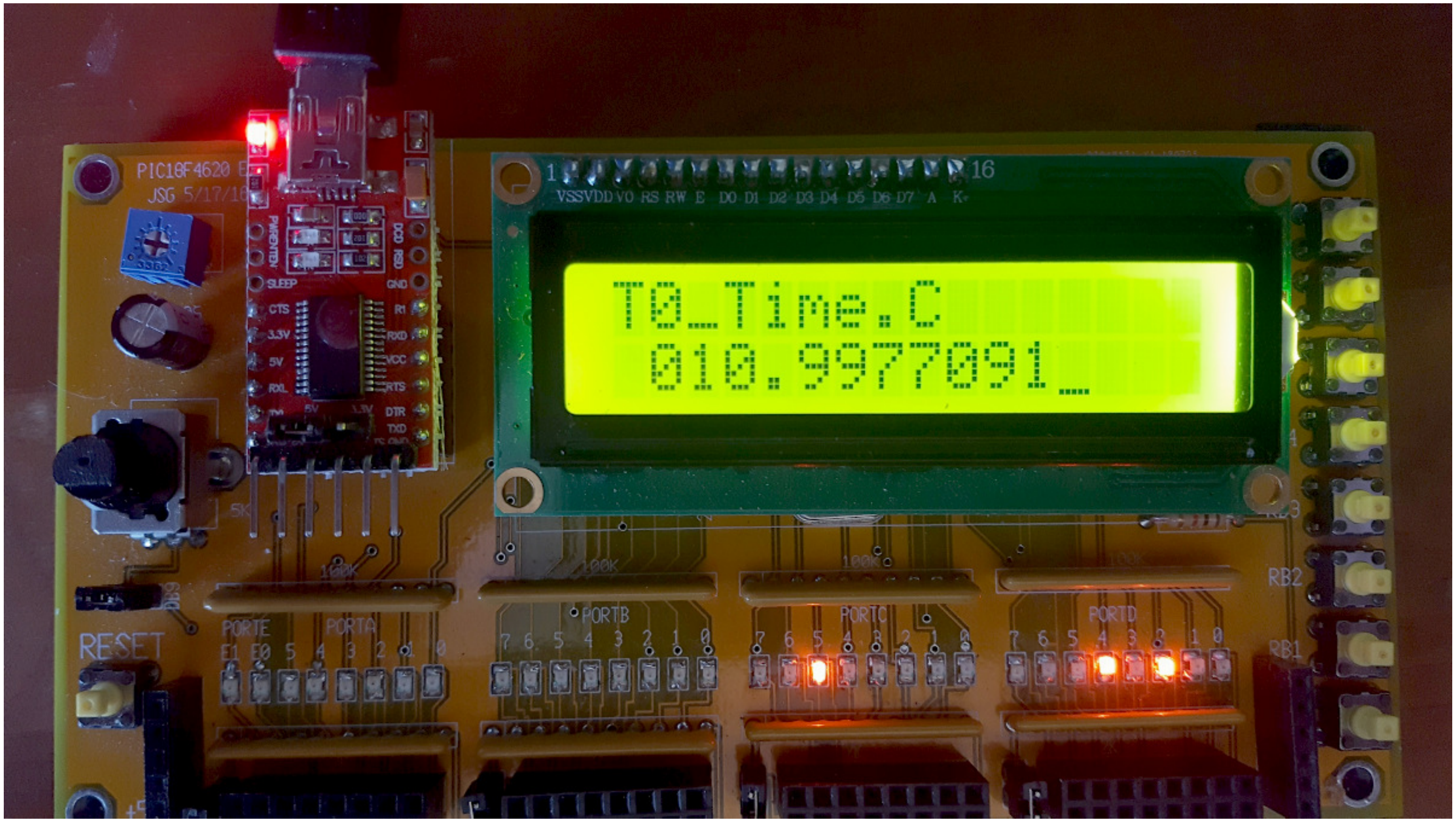
```
// Global Variables
unsigned long int TIME;

// Interrupt Service Routine

void interrupt IntServe(void)
{
   if (TMR0IF) {
      TIME = TIME + 0x10000;
      RC0 = !RC0;
      TMR0IF = 0;
      }
```

Main Routine

```
while(1) {
   LCD_Move(1,0);
   LCD_Out(TIME + TMR0, 10, 7);
   }
```

Result:

- Displaying time in seconds
- With a resolution of 0.000 000 1 second (!)



How small is 100ns?

- Light travels 100ft in 100ns
- Usain Bolt travels 1.044um in 100ns
- Human reflex time is about 1/4 second (2,500,000 clocks)
- Due to relativity, time slows down by 15ns when you fly to London and back

100ns is really really small

- Computers can measure time to an insane degree of accuracy

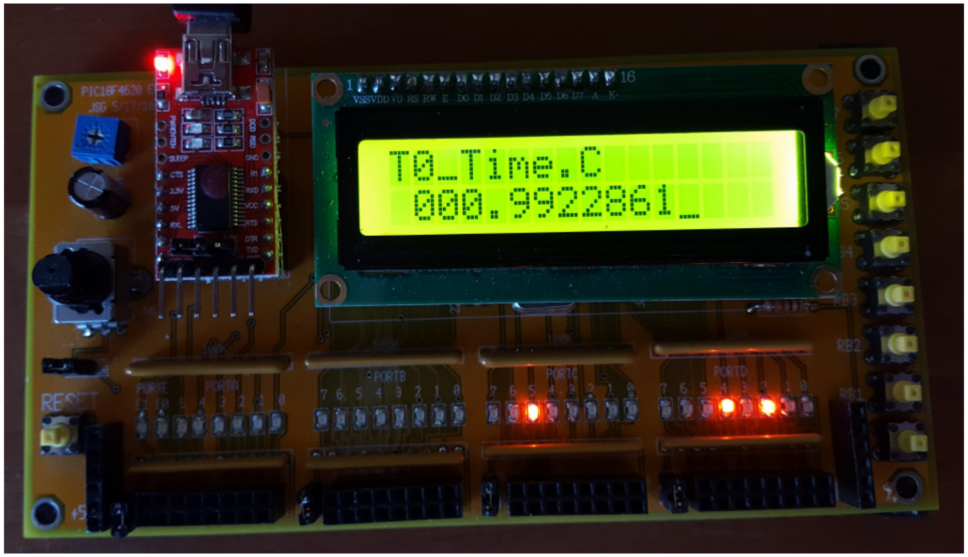
Measure the execution time to 100ns

- Just for fun, determine how long *Wait_ms(1000);* actually takes
- Result = 0.9922861 seconds

Main Routine

```
while(1) {

    TIME0 = TIME + TMR0;
    Wait_ms(1000);
    TIME1 = TIME + TMR0;

    LCD_Move(0,0);
    LCD_Out(TIME1 - TIME0, 7);
    LCD_Out(TIME1 - TIME0, 7);
    }
```

Result



Time for other operations....

- Floating point multiply: 118.3 us
- Cosine(1.2345678) 2.3642 ms
- atan2(x, y) 2.4278 ms
- LCD_Out(Time, 5, 3) 15.5507 ms
- LCD_Out(Time, 10, 7) 16.3106 ms
- Button Press 46.1127 ms